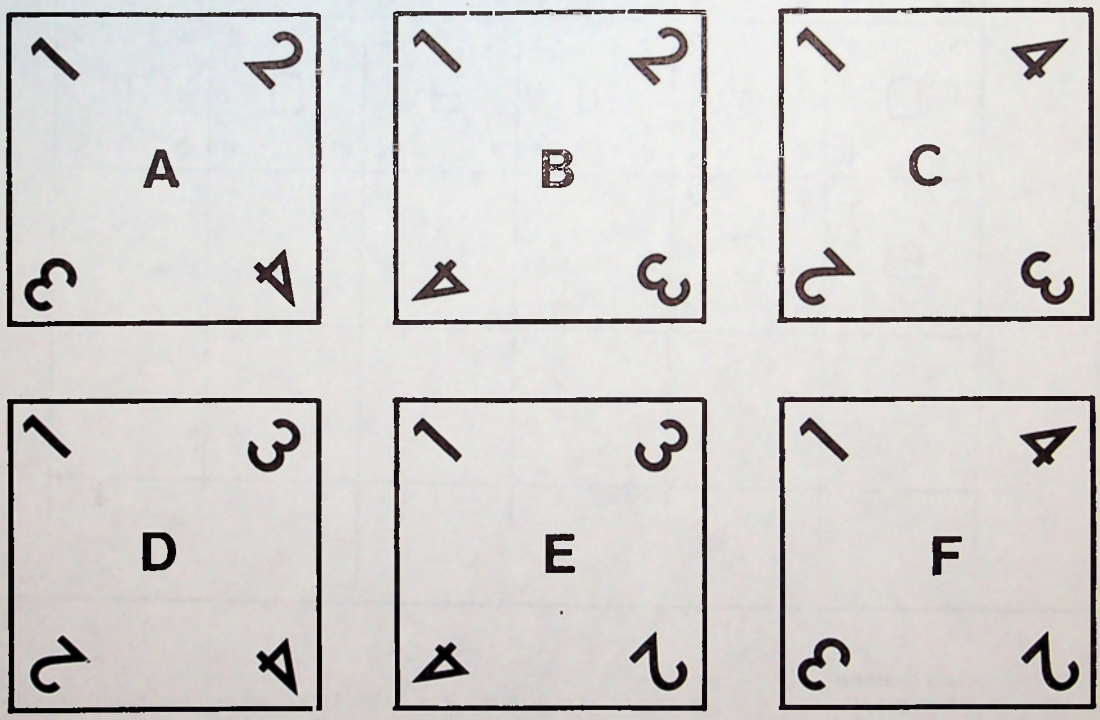
$2.50

58

January 1978 Volume 6 Number 1

Popular Computing

The only magazine devoted to the art of computing

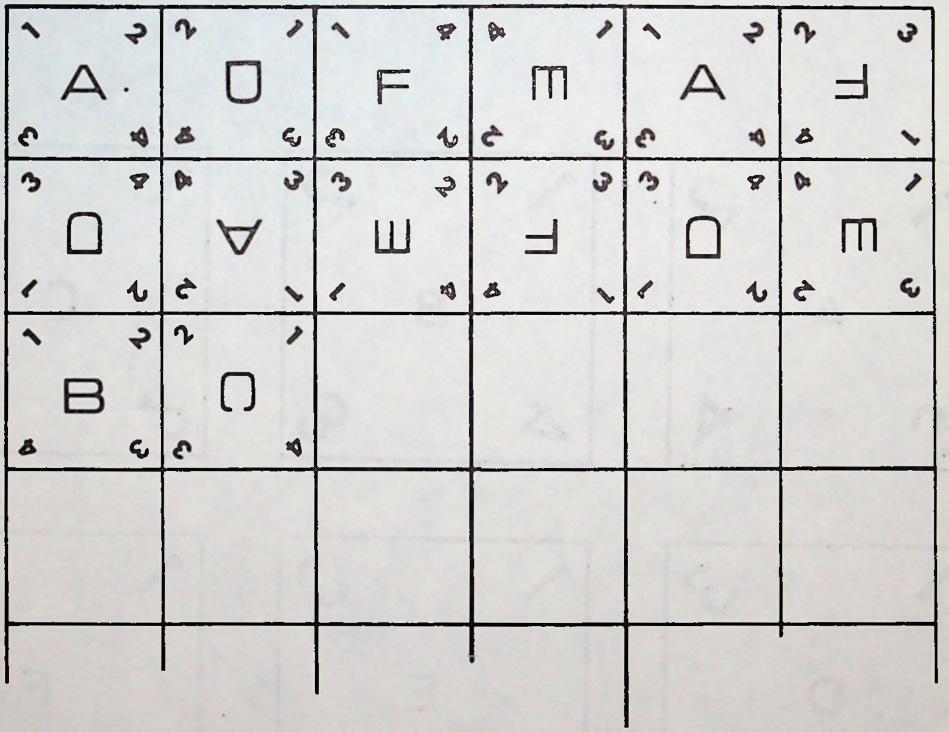


QUADROMINOES

QUADROMINOES

The six squares on the cover contain the possible arrangements of the numbers 1 to 4 at the vertices.

Given six of each of the six types, it should be possible to place them in a 6 x 6 array so that the same number appears at adjacent vertices. The start of such a pattern is shown below, but it has reached an impasse, since no square can fit in the next position.



PROBLEM 219

□

Publisher: Audrey Gruenberger
Editor: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald
Contributing Editors: Richard Andree
William C. McGee
Thomas R. Parkin
Advertising Manager: Ken W. Sim
Art Director: John G. Scott
Business Manager: Ben Moore

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1978 by POPULAR COMPUTING.

SHELL Sorting

The following article is reprinted, with permission, from the July 1959 issue of the *Communications* of the ACM.

Internal sorting is an important topic, and the scheme devised by Don Shell is efficient and simple. The attached flowchart is a modification of the one that appeared in the original article, primarily to cast it in today's notation.

A High-Speed Sorting Procedure

D. L. Shell, General Electric Company, Cincinnati, Ohio

There are a number of methods that have been used for sorting purposes in various machine programs from time to time. Most of these methods are reviewed by Harold Seward [1] in his thesis. One tacit assumption runs through his entire discussion of internal sorting procedures, namely, that the internal memory is relatively small. In other words, the number of items to be sorted is so large that they cannot possibly all fit into the memory at one time.

The methods of internal sorting which he discusses are sorting by:

1) Finding the smallest.
2) Interchanging pairs.
3) Sifting.
4) Partial sort.
5) Merging pairs.
6) Floating decimal sort.

The first four methods all require a time proportional to n^2, where n is the number of items being sorted. The time for the fifth method is proportional to $n(\ln n)$. The time for the sixth method is proportional to $n(\ln r)$, where r is the largest number to be used in a key.

As pointed out in Seward's paper, one would normally choose either method five or six for a rapid internal sort, especially if n is to be very large. The chief drawback of these two methods, however, is the fact that they require twice as much storage as the other four methods.

The advent of very large high-speed random access memories changes the picture relative to sorting somewhat. It is now possible to have a very large number of items to be sorted in memory all at one time. It is highly desirable, therefore, to have a method with the speed characteristics of the merging by pairs and the space characteristics of sifting. If such a method were available it would be possible to sort twice as many items at one time in the machine and still do it at a reasonably high speed.

Such a method is outlined in this paper. The idea is, in fact, to combine some of the properties of merging with some of the properties of sifting. The method is most easily described by reference to the block diagram. Suppose we are given a sequence of elements f_i to be sorted with keys r_i, $i = 1, 2, ..., n$. One begins by dividing the set of elements into n/2 subsets. As can be seen, there will be two elements in each subset, with the possible exception of one which may have three. Each of these subsets is then sorted. It should be noted that each subset of two elements is so placed in the total list that they are separated by approximately n/2 places. Thus, if the two elements are out of sort and must be interchanged they will move a distance of n/2 places in the total list. Once this pass is completed the total set is now divided into approximately n/4 subsets of elements. This, in effect, merges two of the original subsets into one of the new subsets. The merging is not done according to the key, but rather is done by first taking one element from the first subset and one from the second, then another from the first and another from the second, etc. Each of these subsets is then sorted by sifting.

The logic of this whole procedure is outlined in the block diagram. The notation of the block diagram is as follows:

$\longrightarrow$ means "replaces."

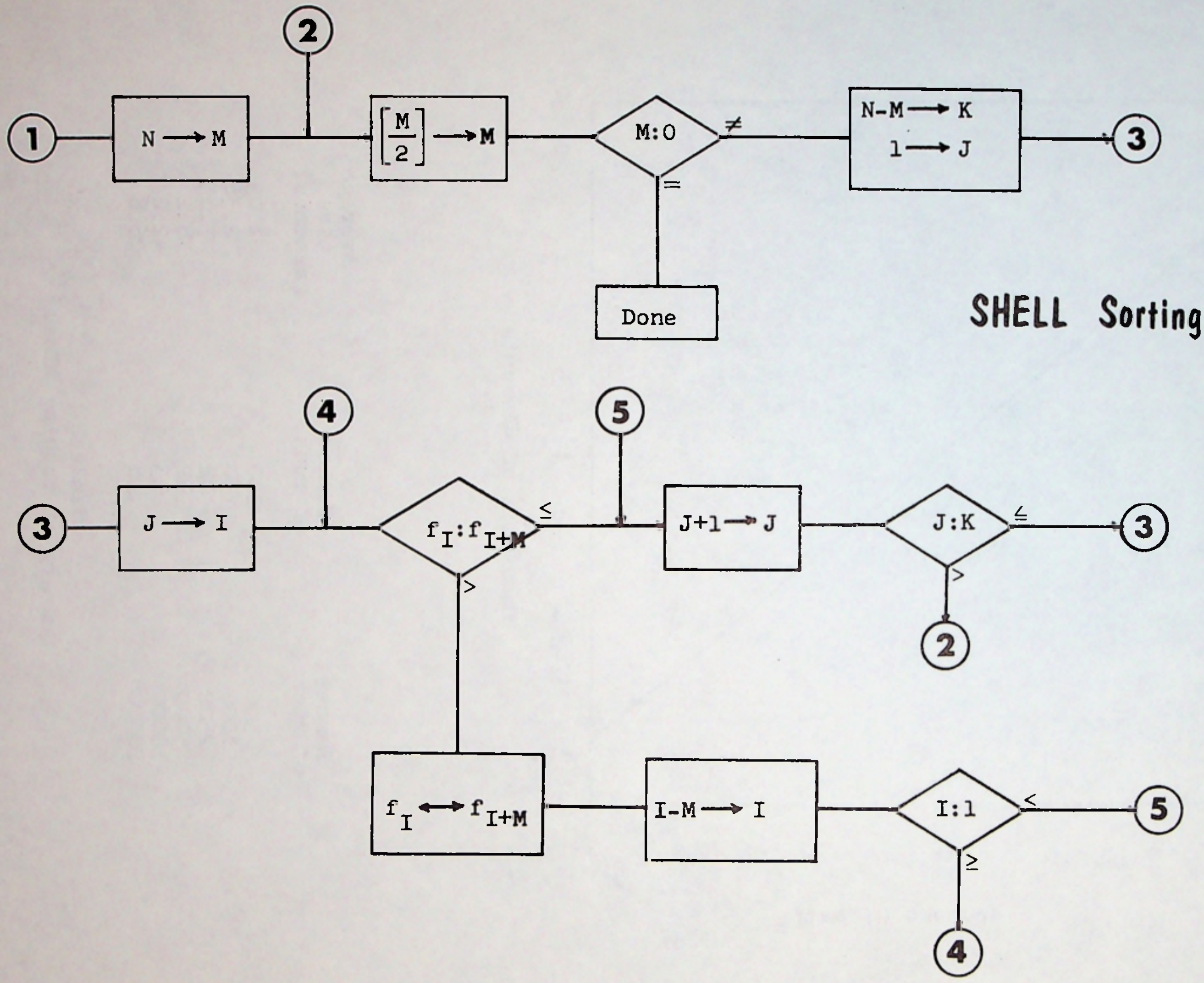
$[m/2]$ means the largest integer less than or equal to.

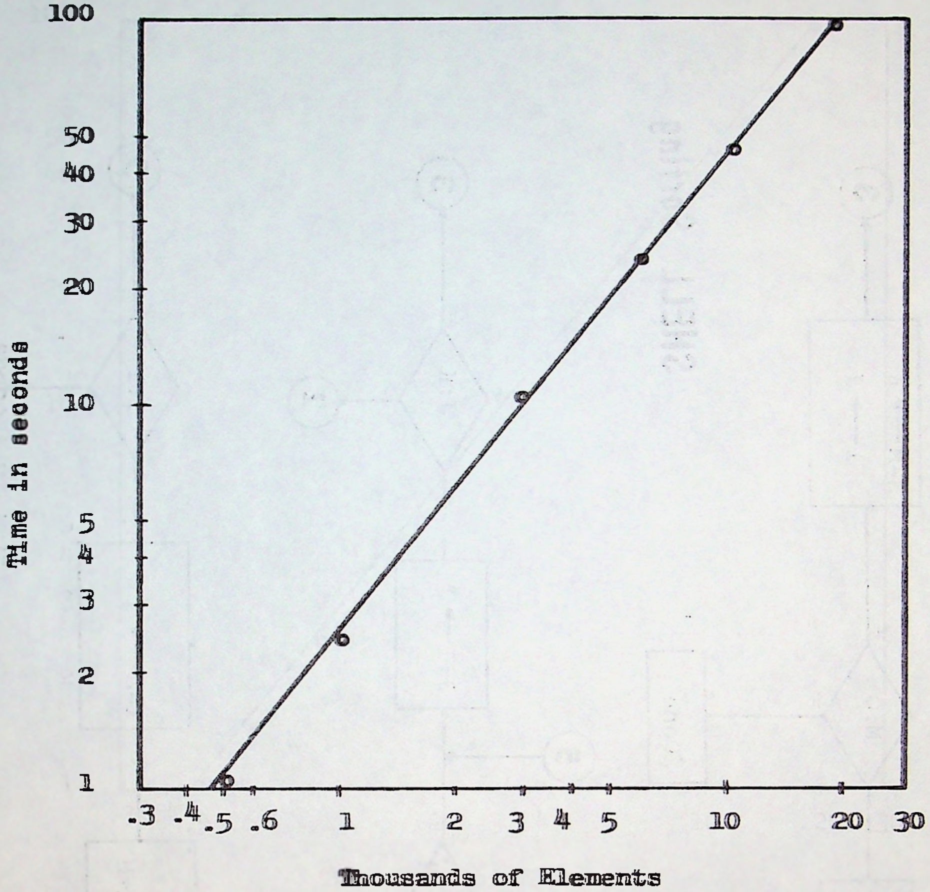
Table 1 illustrates an example sorted by this method. Column 1 is the original arrangement of 11 items. Column 3 is the arrangement after the first complete pass, i.e., with $m = 5$. On the next pass $m = 2$, and column 7 is the result. On the final pass $m = 1$, and the elements are in complete sort.

This particular method requires a negligible amount of storage space in addition to that occupied by the list of items. In addition, it operates at fairly high speed. Thus far, an analytical determination of the expected speed has eluded the writer. However, experimental use has established its speed characteristics. It appears that the time required to sort n elements is proportional to $n^{1.226}$. At least this is true for one-word items which are their own key.

Table 2 shows a summary of average times for sorting numbers of one-word elements which were originally in random order. In this case, $r_i = f_i$. The graph is a plot of the time required to sort a random sequence of <u>n</u> elements with the above program.

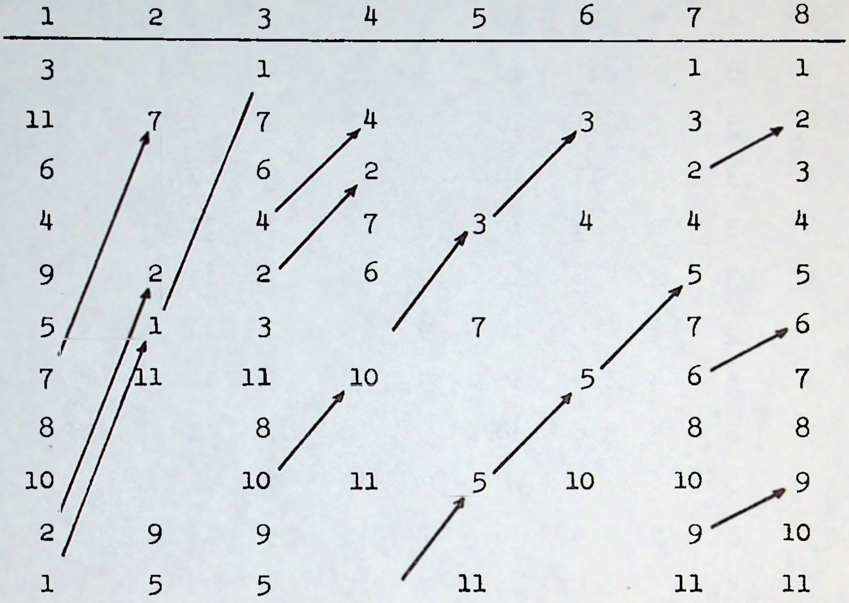
SHELL Sorting





Number of items	Number of runs	Average time (seconds)
500	100	1.03
1000	50	2.36
2500	12	7.85
5000	12	18.2
10000	13	42.2
20000	6	96.8

Table 2
Summary of running times



1	2	3	4	5	6	7	8
3		1				1	1
11	7	7	4		3	3	2
6		6	2			2	3
4		4	7	3	4	4	4
9	2	2	6			5	5
5	1	3		7		7	6
7	11	11	10		5	6	7
8		8				8	8
10		10	11	5	10	10	9
2	9	9				9	10
1	5	5		11		11	11

Table 1

This sorting method is useful in machines with a large high-speed random access memory. The time requirement is only slightly longer than that for merging pairs--but this method can be used to sort twice as many elements in the same memory space. In many cases, the little extra time for internal sorting will be more than compensated by the diminished use of slow memory, such as tapes.

References

1. Seward, Harold H., Information sorting in the application of electronic digital computers to business applications (ASTIA 35462).

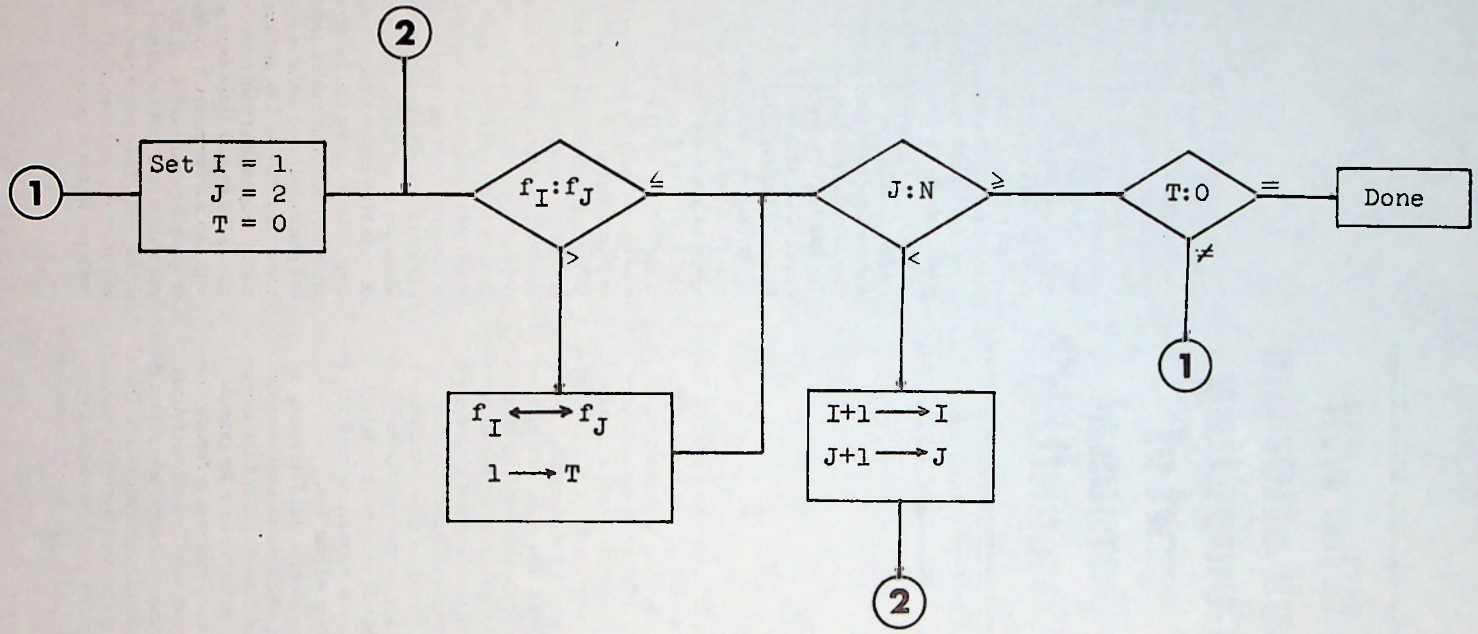
2. Friend, E. H., Sorting on electronic computer systems, J. Assoc. Comp. Mach. 3 (1956), 134.

11	10	9	8	7	6	5	4	3	2	1	← Original ordering
$\underline{6}$	10	9	8	7	$\underline{11}$	5	4	3	2	1	
6	$\underline{5}$	9	8	7	11	$\underline{10}$	4	3	2	1	
6	5	$\underline{4}$	8	7	11	10	$\underline{9}$	3	2	1	
6	5	4	$\underline{3}$	7	11	10	9	$\underline{8}$	2	1	
6	5	4	3	$\underline{2}$	11	10	9	8	$\underline{7}$	1	
6	5	4	3	2	$\underline{1}$	10	9	8	7	$\underline{11}$	
$\underline{1}$	5	4	3	2	$\underline{6}$	10	9	8	7	11	
1	$\underline{3}$	4	$\underline{5}$	2	6	10	9	8	7	11	
1	3	$\underline{2}$	5	$\underline{4}$	6	10	9	8	7	11	
1	3	2	5	4	6	$\underline{8}$	9	$\underline{10}$	7	11	
1	3	2	5	4	6	8	$\underline{7}$	10	$\underline{9}$	11	
1	$\underline{2}$	$\underline{3}$	5	4	6	8	7	10	9	11	
1	2	3	$\underline{4}$	$\underline{5}$	6	8	7	10	9	11	
1	2	3	4	5	6	$\underline{7}$	$\underline{8}$	10	9	11	
1	2	3	4	5	6	7	8	$\underline{9}$	$\underline{10}$	11	

The display above shows the action of Shell sorting on 11 elements originally in descending order, with the arrangement following every interchange.

The following table shows the comparison between bubble sorting (for the worst case) and Shell sorting. The numbers in the table are the number of comparisons and interchanges that are made to effect the complete sort.

Case	Bubble	Shell	Ratio
3	3	3	1.000
4	6	4	.667
5	10	4	.400
6	15	9	.600
7	21	8	.381
8	28	12	.429
9	36	8	.222
10	45	13	.289
11	55	15	.273



For comparison purposes, the flowchart for interchange (or bubble) sorting is given, in the same notation as that used for the Shell scheme.

Who will look after the Computing part of Personal Computing?

by Fred Gruenberger

I guess it didn't astonish me to learn about high fidelity fans who assemble the absolute ultimate in quality equipment but seldom listen to music. I can comprehend the person who tinkers endlessly with high powered engines but does little driving. I can accept that the Wright brothers were not noted for being air travelers. I can even accept the fact that most ham radio operators, who are equipped for superb communication with distant parts of the globe, communicate only the quality of their transmission and never even inquire about the weather conditions in Marrakesh, or wherever they make contact.

But when the hobbyist computer fad struck us a couple of years ago, I fully expected that some of the 20,000 or so people who paid a lot of money and invested hundreds of hours in assembling a machine would then be somewhat interested in computing. I have yet to meet one.

What I have met is a lot of people (including students of mine) who are all bright-eyed and bushy-tailed about their new XYZ computer kit, which they are busy assembling. They all seem to want to impress me with the high ratio of gear acquired to dollars spent, and I *am* impressed because they are about to put together for nearly nothing what would have cost a fortune just a few years back. Being impressed, I inquire gently, "When you get it all assembled and tested, what will you do with it? Usually the answer is "I'll add another 100K bytes of storage, and a CRT display, and a disk drive." So I keep probing: "Yes, and *then* what will you do with it?"

And at this point they all go blank, as though I had raised a completely irrelevant question. They mumble vaguely about maybe writing a super compiler, or a chess playing program, or some such. They never, ever, mention any interest in computing anything.

The experts in the personal computing field ridicule my probing. They cite the example they know, of the person who has indexed his entire record collection on his home computer. (This task, it seems to me, is better served with 3 x 5 cards. I can't quite see the need for millisecond access to a record index. Besides, where is the *computing* in making or using that index?) Or, someone has plans to control every device in his home with his computer. (Again, substitute for a few cheap clock timers. And where is there any *computing* involved?) Each person will, of course, play lots of games with his machine, preferably preprogrammed Star Trek.

The question that I am raising depends, of course, on what someone thinks computing is, and opinions certainly differ. We have, for example, frequently published in *Popular Computing* what we thought were dandy computing problems, only to have them demolished by analytic means. That's fine; if a problem is most expeditiously handled by algebra or calculus, then it is not intelligent to use a computer to solve it. But the dividing line is awfully thin. It should be clear at least that problems that can be solved by staring at the ceiling for a few minutes are not good computer problems. The same ought to be true for problems that are best handled graphically, or with a set of file cards, or by using a punched card sorter.

Such problems are unsuited to computer attack at a low level. At the other extreme, a passable chess playing program is beyond the micro computer; chess is far too complex for miniature machines (not to mention novice programmers or novice chess players). But there are plenty of open-board games that could be attacked with some chance of success (Oware and Fives, for two examples).

I try to keep in mind that it's a free country; if anyone wants to work hard to build a machine whose only function will be to ripple bit patterns through storage, that's fine. But after that, it's time to start using the new gadget for its chief purpose, which is to compute.

The computer-building fad has probably run its course anyway. Fully assembled packaged machines are due out this year at attractive prices, and the days of the soldering iron and wire wrap tool are nearly over. Notice that not too long ago, hobbyists could buy a Heathkit to make a desk calculator, which was lots of fun. Just when many of those $100 kits were completed, their builders noticed that far better machines (factory assembled, tested, and guaranteed) were available at the corner drug store for $39, and Heath quietly dropped that item from their catalog. We are now at the same point with personal computers. Very few people will buy kits and do all the work when better machines are offered already built. The soldering iron fun will be missing, to be replaced by the much greater fun of using a computer.

At first, this fun may take the form of game playing, but eventually (and quite soon) we will see personal machines by the hundreds of thousands, to be used to solve real problems. It is my contention that problem solving by computer will soon become a hobby of its own, and few of its practioners will care much about the internal circuitry of their machine, or even the details of its software, other than to insist that it function properly.

With factory-assembled full-blown computers soon to be available, there should be large numbers of users of personal computers who will have little else to do *but* compute. They will buy the machine for some mundane purpose, like the accounting functions for a small business, and find themselves with lots of computer time left over, at which point they can begin to explore more and more sophisticated problems. Fortunately, there is an unlimited stock of excellent problems waiting to be worked on. Just as an example, let me suggest my favorite.

Suppose there are three numbers, A, B and C, in storage and they are to be arranged in ascending order. The following scheme will do it:

1. Compare A to B; if A is less than or equal to B, do nothing; otherwise interchange A and B.
2. Similarly, compare B and C and interchange if necessary.
3. Similarly, compare A and B and interchange if necessary.

Let's express all this logic with this notation: AB BC AB. If, now, we have four numbers to sort, the logical extension of the above scheme is: AB BC CD AB BC AB. Many textbooks have stated that that *is* the proper algorithm; namely, *six* comparisons and interchanges to sort four numbers. The scheme does indeed work, but it can be done with only *five* comparisons and interchanges: AB CD AC BD BC.

So that brings us to the problem of sorting five numbers by direct internal sorting (as opposed to merge sorting, the Shell sorting algorithm or bubble sorting). We have arrived at this table:

Numbers to be sorted	Comparisons in theory	Comparisons actually needed
2	1	1
3	3	3
4	6	5
5	10	?

The extension of the usual theory tells us that the scheme for sorting five numbers should be: AB BC CD DE AB BC CD AB BC AB and that scheme will work. But since we know that, for four numbers only five comparisons and interchanges are needed, we know that for five numbers we will need less than ten (but more than five) comparisons. The possible number of comparisons is six, seven, eight or nine – and no one knows which. Except for trying all possible combinations (and the number of them is enormous), we don't even know how to go about it. Now, *that's* a computing problem. It is readily understood (that is, it is well defined); it involves only a few small numbers; and anyone could attack it, using simple equipment. Perhaps it could be done analytically, but no method is apparent. (Note: Sorting theory indicates that the number of comparisons needed to sort K things goes up be $\log_2$ K so that the answer to the problem I've posed is 8. I'll accept that; what is left is determining just *what* 8 comparisons will do the job.)

There is an unending list of good problems to be worked on, in geometry, number theory, and combinatorial work. No one knows the number of ways a 12 x 12 checkerboard can be cut into four congruent pieces (following the lines of the board). No one knows how many square polyominoes of length 19 squares there are. Not much is known about the behavior of random processes. All of these problem situations are wide open for exploration by personal computer users.

Besides these vast areas of the unknown, there are also countless problems that *have* been solved, but solving them again (perhaps in new ways) provides insight into the computing art. Consider as an example one of the problems we used as a contest in *Popular Computing*, which got to be known as the Take/Skip problem: Start with all the positive natural numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, . . . Take the first number; reject the second; take the third; and so on. This is stage 1, and it leaves us with the odd integers.

Out of the numbers remaining at stage 1, in stage 2 take two numbers; reject two; take two; and so on. We now have left: 1, 3, 9, 11, 17, 19, 25, 27, 33, 35, 41, . . .

This process is continued indefinitely. After stage 3 (take 3; reject 3; and so on) we would have left: 1, 3, 9, 25, 27, 33, 49, 51, 57, 73, 75, . . . and the Problem is: What numbers will survive all of the stages?

Several of us tried all sorts of approaches to this problem. As I recall, we wrote collectively some seven different programs, one of which (after several *hours* of running on a moderately fast machine) yeilded 42 numbers. The contest winners wrote a program that yielded (in some *minutes* of CPU time) 1200 numbers. It's a dandy problem to sharpen your computing wits on.

Very little is known about how to solve problems, and particularly how to solve computing problems. You can't reduce problem solving to a set of rules, any more than you could construct rules for writing great symphonies or plays. But we have observed this: those who are outstanding at problem solving have done a great deal of it. In the belief then that "The way to learn computing is to compute," the thing to do with a computer is find some problem area that you enjoy and plunge in and solve a lot of problems. Besides being fascinating, it's the way to learn the art. It is probably the only way.

When it comes to *computing*, there are four things to be learned:

What constitutes a computer problem. (As opposed to trivia, or problems done better by other means, or problem solutions that would take a lifetime of the largest and fastest machine in existence.)

How to solve a computer problem. (This includes not only all the subject's mechanics, but much of the same low cunning used to solve problems in any milieu.)

How to tell that you've computed correctly. (This is the tough one. The printed output from our machines always looks authentic, but it is a high form of art to gain some assurance that what you've computed is not garbage.)

What things are worth computing. (This topic is subjective and personal – no two people would agree on the criteria for worthwhileness. But each person should decide for himself on what he regards as worth doing.)

The whole game is fascinating and addictive, and the excitement of it can last for over 30 years. As a hobby, it will probably get more exciting as it builds up to a national craze.

We have already passed the crossover point at which man/machine roles reverse themselves. Not too long ago, whether we liked it or not, people worked to serve the computers. In order to have any computing power at all, someone had to invest a million dollars or so. Then, to protect that investment, elaborate mechanisms had to be set up to keep the machines occupied around the clock. The results were intended to serve people, to be sure, but a great many people spent all their time serving the machines.

Beginning in 1968 or so, this inverse symbiosis began to undergo remission, and the new trend has accelerated rapidly since 1975. Computing power has become so cheap and so portable that it is now clear that we can bring the machine to the problem (which is as good a definition of a mini computer as you're going to get) and, in the extreme, bring the machine to the individual; that is, personal computing. A natural consequence of this trend (and one that is difficult for many old-timers to grasp) is that a lot of equipment is going to be idle much of the time; this is precisely the price we pay for convenience.

At the same time that computers are becoming available (in price and physical size) to fit the problems, calculators (which are a different breed of cat) are also becoming cheap and powerful. I do not believe that these lines of machines are going to cross. Programmable calculators will continue to get more powerful, but will still offer decimal programming and built-in functions (e.g., logarithms, trig functions, and statistical functions).

Each individual user will decide for himself whether to acquire a computer or a high powered calculator (but notice that a computer can be readily programmed to duplicate any of the actions of a calculator). Either way, it's clear that this country is going to be flooded with personal computing devices of ever-increasing power. The bulk of them will become dedicated to one or two quite mundane tasks, but for some the wonderful world of problem solving with computers will suddenly open up.

There will really be only one small problem left: whatever machine one buys, there will always be the nagging fact that if you had only waited one more month. . . ■

Reprinted by permission of
Personal Computing
September/October 1977

In figures I searched to discover
A sequence to mystery's knot.
Great logarithm! Nothing will recur
In thy realm but entity naught.

The author of this most ingenious mnemonic is unknown. The number of letters in each word give 22 digits of the natural logarithm base, e. The mnemonic rhymes, expresses a truth about logarithms, and solves the "zero" problem of mnemonics by the use of "nothing" and "naught."

SETS OF ROOTS

PROBLEM 220

The sum:

$$\sqrt{1}+\sqrt{2}+\sqrt{3}+\sqrt{4}+\sqrt{5}+\sqrt{6}$$

exceeds the limit of 10 (an arbitrary limit). The sum:

$$\sqrt{7}+\sqrt{8}+\sqrt{9}+\sqrt{10}+\sqrt{11}$$

exceeds the limit of 12, which is twice the largest N of the first sum. The sum:

$$\sqrt{12}+\sqrt{13}+\sqrt{14}+\sqrt{15}+\sqrt{16}+\sqrt{17}$$

exceeds the limit of 22, which is twice the largest N of the second sum.

Each successive sum contains the integers that follow in sequence from the last value of the preceding sum, and proceeds to a total that is twice the value of the last N. We are developing this table:

Stage	Limit	Last N in the sum
1	10	6
2	12	11
3	22	17
4	34	25
5	50	35
6	70	46
7	92	59
8	118	74
9	148	91
10	182	110

Problem: What values of N will be involved in the 1000th stage; what will be their sum; and what will be the limit for that stage?

There is undoubtedly a true answer to this problem, and that answer can probably only be obtained by computer. However, the true answer may require taking the square roots to very high precision; it may be that with limited precision, a false answer is obtained.

□

Problem Solution

Problem 198, Wind Chill, in issue 55, called for evaluating the Weather Bureau's equation for equivalent temperatures. Their equation is given in terms of Celsius temperatures and wind speeds in meters/second. The problem called for constructing a table, with wind speeds in miles per hour from 5 to 45 and temperatures in Fahrenheit degrees from -45 to +45. The construction of the table makes a nice coding exercise.

William Bourn, Glastonbury, Connecticut, constructed the table given here, using a 6-line program in APL. His results have been rounded to one decimal place.

Mr. Bourn also produced some results for Problem 193 (Throwback). Given all the natural numbers starting with 3, each number at the head of the stream is to be thrown back the number of positions indicated by its value (this was illustrated on the cover of issue 55). The problem was to determine the number of moves needed for each new larger number to appear at the head of the stream. With another APL program, Mr. Bourn extended the given table as follows:

4	1
5	2
6	3
7	5
8	7
9	10
10	14
11	19
12	26
13	35
14	47
15	63
16	85
17	114
18	153
19	205
20	274
21	366
22	489
23	653
24	871
25	1162
26	1550
27	2067
28	2757
29	3677
30	4903
31	6538
32	8718
33	11625
34	15501
35	20669
36	27559
37	36746
38	48995
39	65327
40	87103
41	116138
42	154851
43	206469
44	275293

[These results were extended by our staff to include the values 367058, 489411, 652549, and 870066.]

Perhaps it is improper to reach a conclusion from only 45 data points, but every value on the list conforms to the rule:

$$\text{next} = \left[4/3 \text{ times last} + 1\right]$$

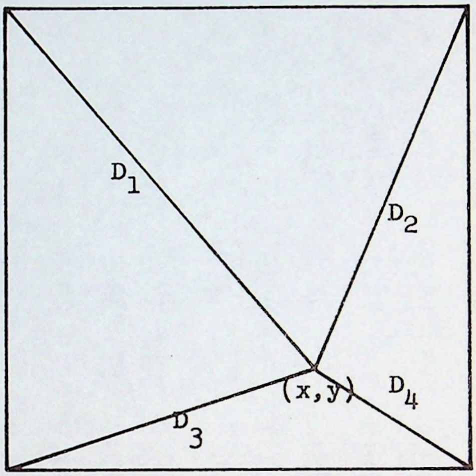
If this rule holds, then the number 100 will first appear after 3.641718071 E12 moves. □

Wind speeds in mph

Temperature in degrees F	5	10	15	20	25	30	35	40	45
-45	-52.0	-76.5	-92.1	-103.0	-111.1	-117.0	-121.3	-124.4	-126.4
-40	-46.7	-70.3	-85.4	- 95.9	-103.6	-109.3	-113.5	-116.5	-118.4
-35	-41.5	-64.2	-78.6	- 88.8	-96.2	-101.7	-105.7	-108.5	-110.4
-30	-36.2	-58.0	-71.9	- 81.7	-88.8	- 94.1	- 97.9	-100.6	-102.5
-25	-31.0	-51.9	-65.2	- 74.5	-81.4	- 86.4	- 90.1	- 92.7	- 94.5
-20	-25.7	-45.7	-58.5	- 67.4	-73.9	- 78.8	- 82.3	- 84.8	-86.5
-15	-20.5	-39.6	-51.7	- 60.3	-66.5	- 71.1	- 74.5	- 76.9	- 78.5
-10	-15.2	-33.4	-45.0	- 53.1	-59.1	- 63.5	- 66.7	- 69.0	- 70.5
- 5	- 9.9	-27.3	-38.3	- 46.0	-51.7	- 55.9	- 58.9	- 61.1	- 62.5
0	- 4.7	-21.1	-31.5	- 38.9	-44.3	- 48.2	- 51.1	- 53.2	- 54.6
5	.6	-15.0	-24.8	- 31.8	-36.8	- 40.6	- 43.3	- 45.3	- 46.6
10	5.8	- 8.8	-18.1	- 24.6	-29.4	- 32.9	- 35.5	- 37.4	- 38.6
15	11.1	- 2.6	-11.4	- 17.5	-22.0	- 25.3	- 27.7	- 29.5	- 30.6
20	16.3	3.5	- 4.6	- 10.4	-14.6	- 17.7	- 19.9	- 21.5	- 22.6
25	21.6	9.7	2.1	- 3.3	- 7.2	- 10.0	- 12.1	- 13.6	- 14.6
30	26.9	15.8	8.8	3.9	.3	- 2.4	- 4.3	- 5.7	- 6.6
35	32.1	22.0	15.5	11.0	7.7	5.2	3.5	2.2	1.3
40	37.4	28.1	22.3	18.1	15.1	12.9	11.3	10.1	9.3
45	42.6	34.3	29.0	25.3	22.5	20.5	19.1	18.0	17.3

Problem Solution

Problem 211 (A Problem of Scale) in issue 56 called for locating the point (x,y):



so that the function

$$F = D_1 + D_2^2 + D_3^3 + D_4^4$$

would be a minimum. For a square of any given size, K, the point (x,y) is uniquely located. For some value of K, the point will have x = y, and Problem 211 was to find that value of K. The following analysis of the problem is by Herman P. Robinson, Lafayette, California.

$$D_1 = \sqrt{x^2 + (K - y)^2}$$

$$D_2^2 = (K - x)^2 + (K - y)^2$$

$$D_3^3 = (x^2 + y^2)^{3/2}$$

$$D_4^4 = \left[(K - x)^2 + y^2\right]^2$$

To obtain the minimum value of F for any K, set the partial derivatives of F with respect to x and y equal to zero:

➡

$$\frac{x}{\sqrt{x^2 + (K - y)^2}} - 2(K - x) + 3x\sqrt{x^2 + y^2}$$

$$- 4(K - x)\left[(K - x)^2 + y^2\right] = 0$$

$$\frac{-(K - y)}{\sqrt{x^2 + (K - y)^2}} - 2(K - y) + 3y\sqrt{x^2 + y^2}$$

$$+ 4y\left[(K - x)^2 + y^2\right] = 0$$

The first of these equations can be solved for x and the second for y by using the Newton-Raphson procedure.

Now we need more information. On page 5 of issue No. 56, there is a table giving x and y for various values of K. I have added another column h = x - y:

K	x	y	h = x - y
.88	.44358538	.44951220	-.00592682
.89	.45032661	.45078190	-.00045529
.90	.45704920	.45202660	+.00502260

So, the value of dK/dh around K = .89 is approximately

$$\frac{.90 - .88}{.00502260 - (-.00592682)} = 1.826581 .$$

Now a new estimate for K can be obtained:

$$K(\text{new}) = .89 - 1.826581(-.00045529) = .89083162.$$

This is the Newton-Raphson method. Using this new value of K we can calculate a new set of x and y values, which will produce a still better K, using the original value of dK/dh. By many iterations, I find the following values:

$$K = .8908315315\ 1652480432\ 4995504768\ 8739381130\ 4089237299$$

$$x = y = .4508863333\ 7101091010\ 0287292081\ 1989834723\ 9716025908$$

$$D_1 = D_4 = .6299605249\ 4743658238\ 3605303639\ 1141752851\ 2573235075$$

$$D_2 = .6221764659\ 1830437384\ 4184951849\ 5846874115\ 8062890960$$

$$D_3 = .6376495677\ 4196026526\ 6860875233\ 6715325697\ 0641442521$$

$$F = 1.4338205938\ 8422501041\ 3051987171\ 1031122291\ 6138691326$$

Our first new estimate of K was very good, suggesting that David Babcock did a good job. One very interesting relationship appeared:

$$\left[x^2 + (K - x)^2\right]^{-3/2} = 4$$

from which $K = x + \sqrt{4^{-2/3} - x^2}$

This can be confirmed by putting this expression for K into the equations obtained for the partial derivatives (and also x = y). This result gives these expressions for the D's:

$$D_1 = D_4 = 4^{-1/3}$$

$$D_2 = \sqrt{2 \cdot 4^{-2/3} - 2x^2}$$

$$D_3 = x\sqrt{2}$$

$$F = 4^{-1/3}(5/4 + 2 \cdot 4^{-1/3}) - 2x^2 + x^3\sqrt{8}$$

HIDE and SEEK

A block of 36 words of storage contains this information:

227	318	311	143	101	353	325	164	150	115	304	283
234	206	346	332	290	136	220	339	297	122	178	199
213	262	108	269	255	185	129	248	276	171	192	157

which is a scrambled arrangement of the numbers in the progression 101, 108, 115, 122,...,353 with one element missing. What would be an efficient scheme for finding the missing element?

This is intended as a computing problem. It occurs in the course of a larger problem; the block of storage must be examined to determine the missing element, and the nature of the sequence of elements is known (in the example, it is a simple arithmetic progression).

The goal of efficiency is taken in at least two ways: first, in terms of storage capacity, and second in terms of CPU time.

In each of the following cases, the block(s) of storage are to be taken as having 10,000 words, rather than the 36 words in the example.

I. The numbers in a block are in arithmetic progression and each time the block is to be searched for the missing number the parameters of the progression are known.

II. The numbers in a block are successive values of a quadratic (or other) function. For example, they are values of $x^2/2 + x/2 + 101$ for $x = 0, 1, 2, \ldots, 10001$; that is, the numbers 101, 102, 104, 107, 111,...,50015102, with one term missing.

III. The scrambled numbers in a block are formed as an ascending sequence (such as 101, 108, 118, 127, 130,...) with no difference between successive terms exceeding 10. The missing term forms a gap of more than 10 but not exceeding 20.

IV. The numbers in a block are formed according to a scheme like this: start with 101 and add 6 when a prime appears; otherwise, add 2. Thus, the generating sequence could be: 101, 107, 113, 119, 121, 123, 125, 127, 133,...

V. There are <u>two</u> blocks of 10,000 words each, generated as 5-digit random numbers, and exactly one number is duplicated between the two blocks. Find the duplicate number.

VI. As in V, there are two blocks of 10,000 5-digit numbers, and exactly one number is <u>not</u> duplicated between the blocks. Find the uncommon number.

For each of these situations, many schemes suggest themselves. For example, the given numbers could be <u>sorted</u>, but that approach is costly in terms of CPU time. Or, the given numbers could be matched against a duplicate block of numbers, but that approach is costly in terms of storage space.

For case I, Associate Editor David Babcock suggests an ingenious attack. Express each number in the block mod 3, 5, 7, 11, 13, and so on, and sum these modular values in separate counters. The counter values can then be used to deduce the missing number, using the Chinese Remainder theorem.

We invite discussion of other possible attacks on these problems.

□

```
122013682599111006870123878542304692625357434280319284219241
358838584537315388199760549644750220328186301361647714820358
416337872207817720048078520515932928547790757193933060377296
085908627042917454788242491272634430567017327076946106280231
045264421887878946575477714986349436778103764427403382736539
747138647787849543848959553753799042324106127132698432774571
554630997720278101456108118837370953101635632443298702956389
662891165897476957208792692887128178007026517450776841071962
439039432253642260523494585012991857150124870696156814162535
905669342381300885624924689156412677565448188650659384795177
536089400574523894033579847636394490531306232374906644504882
466507594673586207463792518420045936969298102226397195259719
094521782333175693458150855233282076282002340262690789834245
171200620771464097945611612762914595123722991334016955236385
094288559201872743379517301458635757082835578015873543276888
868012039988238470215146760544540766353598417443048012893831
389688163948746965881750450692636533817505547812864000000000
000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000
```

Factorial 500